Lecture7(Part2)
Chapter8

Topics covered:
Pipelining

Instruction hazards

- Instruction fetch units fetch instructions and supply the execution units with a steady stream of instructions.
- If the stream is interrupted then the pipeline stalls.
- Stream of instructions may be interrupted because of a cache miss or a branch instruction.

Instruction hazards (caused by unconditional Branch)

Consider a two-stage pipeline,
first stage is the instruction fetch stage and the second stage is the instruction execute stage.
Instructions I_1, I_2 and I_3 are stored at successive memory locations.
I_2 is a branch instruction with branch target as instruction I_k.
I_2 is an unconditional branch instruction.

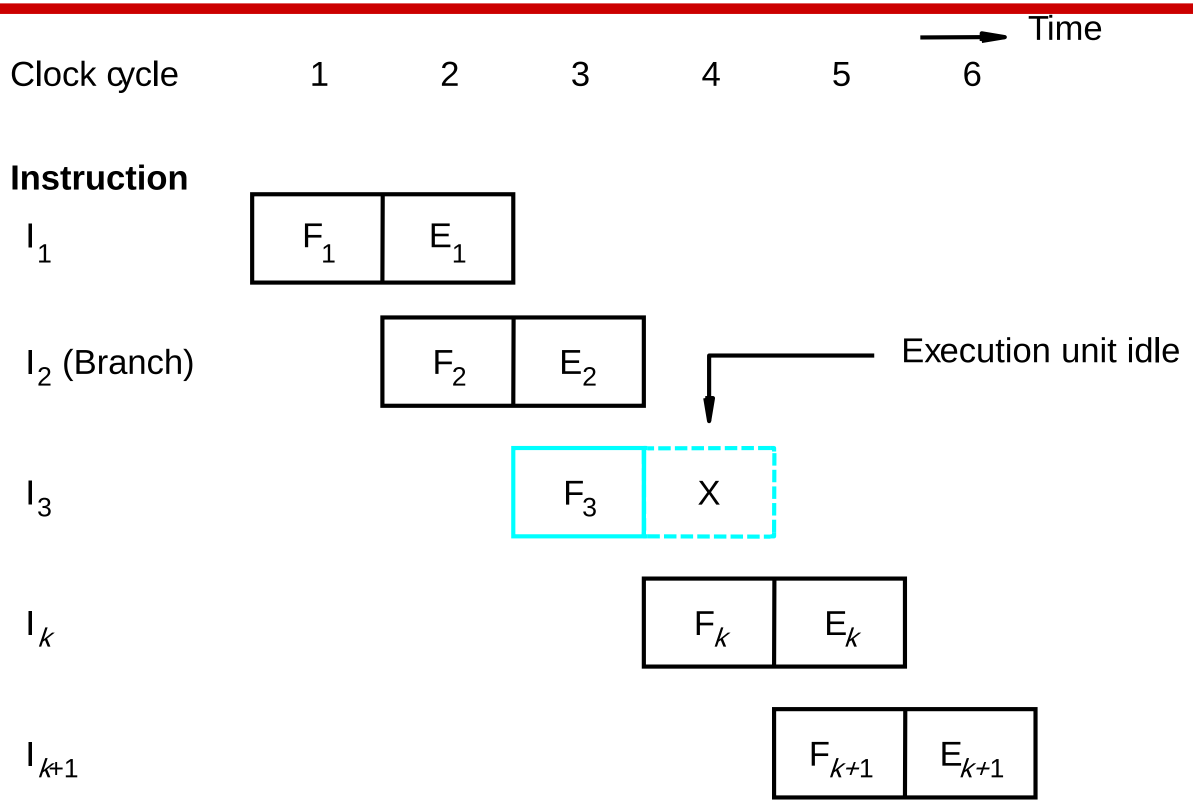
Clock cycle 3:

- *Fetch unit is fetching instruction I_3.*
- *Execute unit is decoding I_2 and computing the branch target address.*

Clock cycle 4:

- *Processor must discard I_3 which has been incorrectly fetched and fetch I_k.*
- *Execution unit is idle, and the pipeline stalls for one clock cycle.*

Instruction hazards (Example)



An idle cycle caused by a branch instruction.

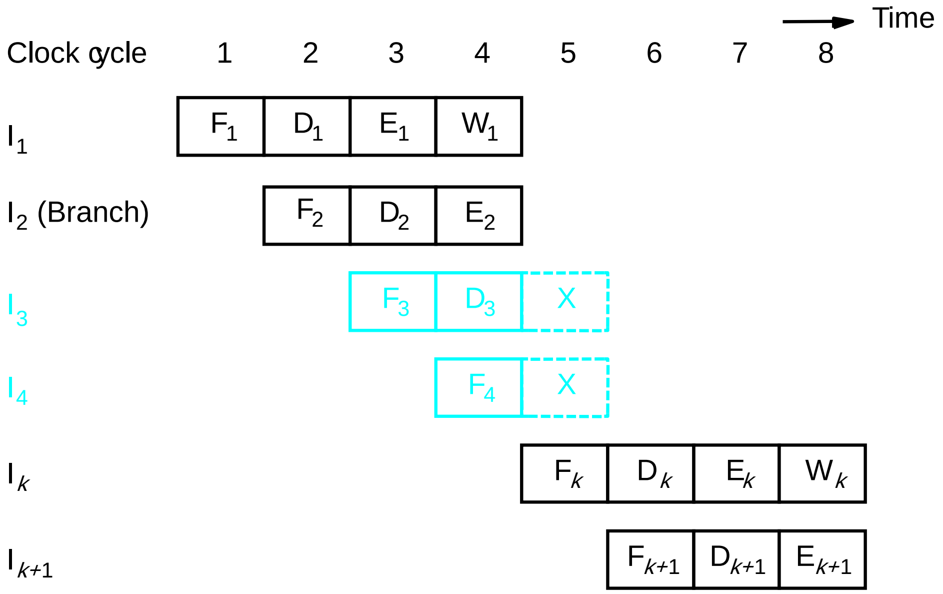
- *Pipeline stalls for one clock cycle.*
- *Time lost as a result of a branch instruction is called as* ***branch penalty****.*
- *Branch penalty is one clock cycle.*

Instruction hazards (contd..)

Branch penalty depends on the length of the pipeline, may be higher for a longer pipeline.

For a four-stage pipeline:

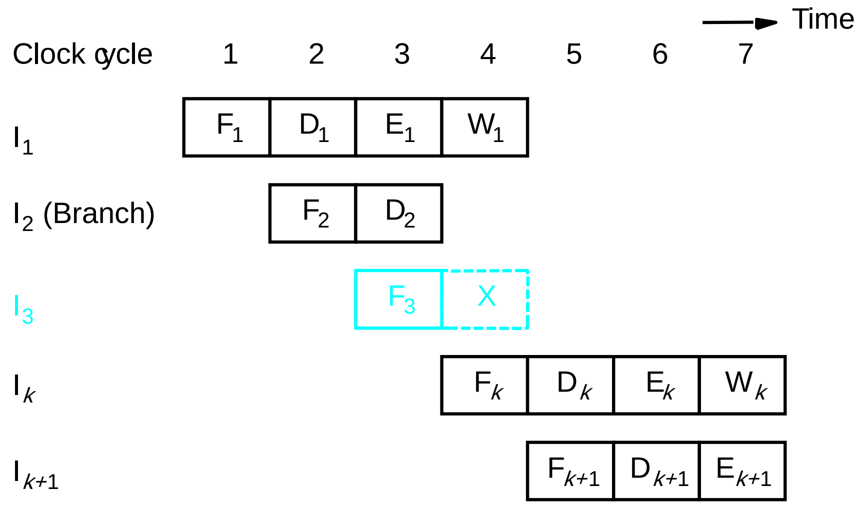
- *Branch target address is computed in stage E2.*
- *Instructions I3 and I4 have to be discarded.*
- *Execution unit is idle for 2 clock cycles.*
- *Branch penalty is 2 clock cycles.*



(a) Branch address computed in Execute stage

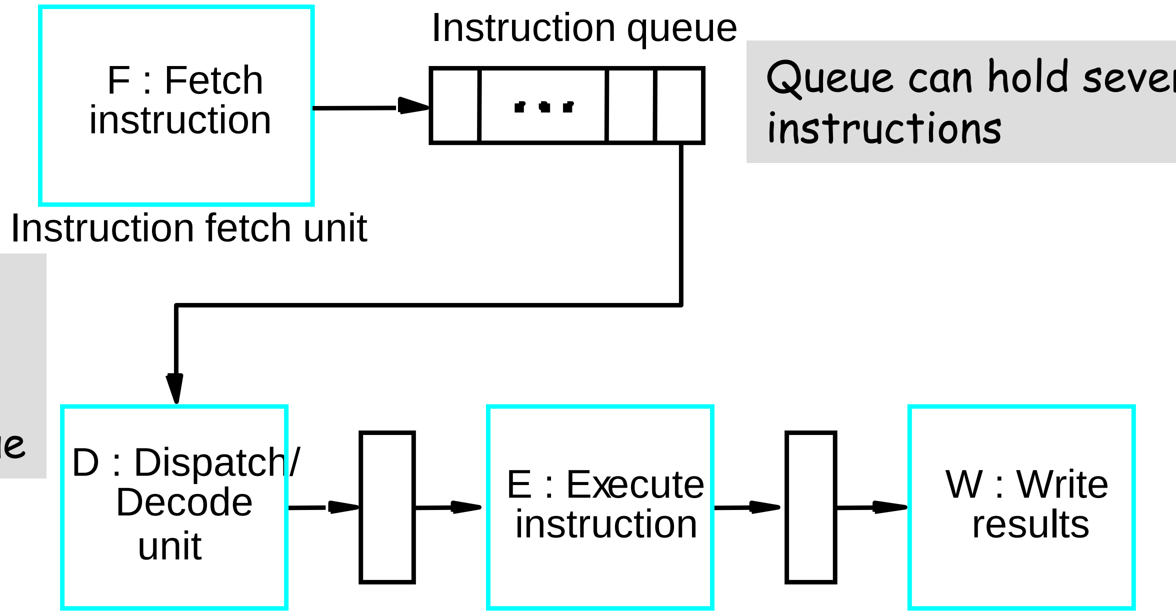
Instruction hazards solution (contd..)

- ***Branch penalty can be reduced by computing the branch target address earlier in the pipeline.***
- *Instruction fetch unit has **special hardware** to identify a branch instruction after the instruction is fetched.*
- *Branch target address can be computed in the Decode stage (D_2), rather than in the Execute stage (E_2).*
- *Branch penalty is only one clock cycle.*



(b) Branch address computed in Decode stage

Instruction hazards solution (Instruction Queue and Prefetching)



Queue can hold several instructions

Fetch unit fetches instructions before they are needed & stores them in a queue

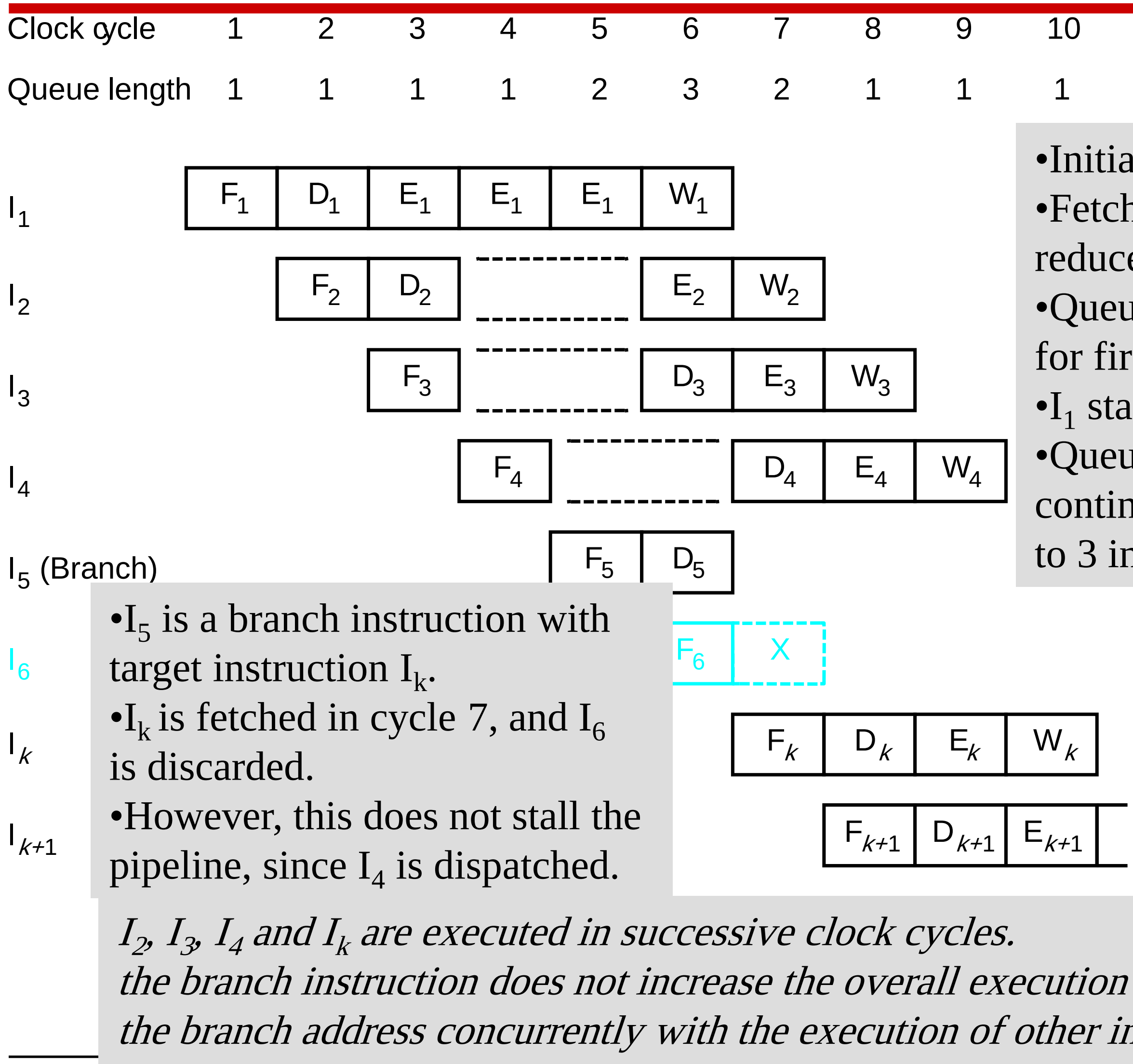
Dispatch unit takes instructions from the front of the queue and dispatches them to the Execution unit. Dispatch unit also decodes the instruction.

Instruction Queue and Prefetching

Instruction hazards (contd..)

- Preceding organization to be effective, Fetch unit must have sufficient decoding and processing capability to recognize and execute branch instructions.
- It attempts to keep the instruction queue filled at all times to reduce the impact of occasional delays when fetching instructions.
- If pipeline stalls because of a data hazard:
  - Dispatch unit cannot issue instructions from the queue.
  - Fetch unit continues to fetch instructions and add them to the queue.
- If there is a delay in fetching because of a cache miss or a branch:
  - Dispatch unit continues to dispatch instructions from the instruction queue.

Instruction hazards (contd..)


Clock cycle 1 2 3 4 5 6 7 8 9 10
Queue length 1 1 1 1 2 3 2 1 1 1
I_1: F_1 D_1 E_1 E_1 E_1 W_1
I_2: F_2 D_2 E_2 W_2
I_3: F_3 D_3 E_3 W_3
I_4: F_4 D_4 E_4 W_4
I_5 (Branch): F_5 D_5
I_6: F_6 X
I_k: F_k D_k E_k W_k
I_{k+1}: F_{k+1} D_{k+1} E_{k+1}

- Initial length of the queue is 1.
- Fetch adds 1 to the queue, dispatch reduces the length by 1.
- Queue length remains the same for first 4 clock cycles.
- I_1 stalls the pipeline for 2 cycles.
- Queue has space, so the fetch unit continues and queue length rises to 3 in clock cycle 6.

- I_5 is a branch instruction with target instruction I_k.
- I_k is fetched in cycle 7, and I_6 is discarded.
- However, this does not stall the pipeline, since I_4 is dispatched.

I_2, I_3, I_4 and I_k are executed in successive clock cycles.
the branch instruction does not increase the overall execution time because Fetch unit computes the branch address concurrently with the execution of other instructions.
This is called as branch folding.

Instruction hazards (contd..)

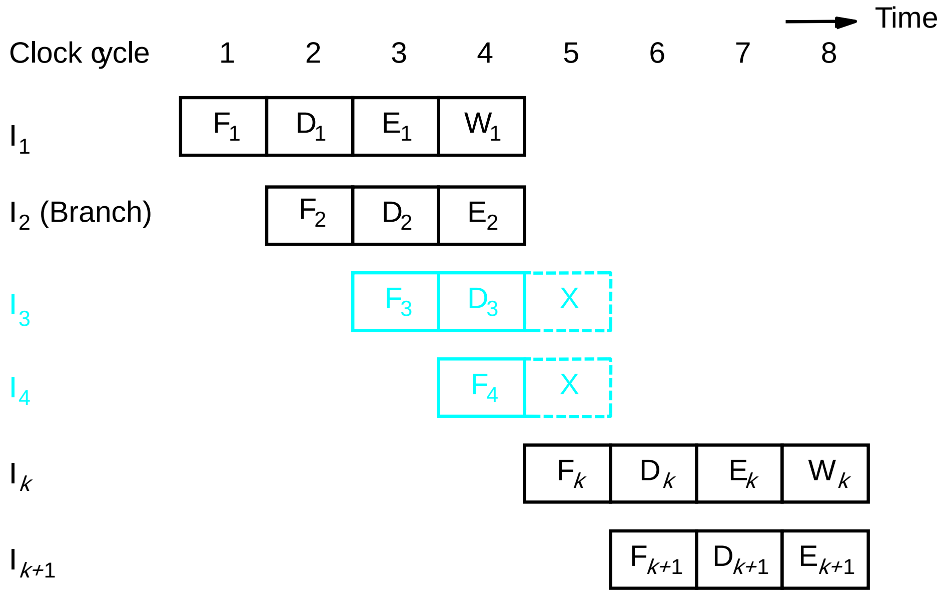
- Branch folding can occur if there is at least one instruction available in the queue other than the branch instruction.
  - Queue should ideally be full most of the time.
  - Increasing the rate at which the fetch unit reads instructions from the cache.
  - Most processors allow more than one instruction to be fetched from the cache in one clock cycle.
  - Fetch unit must replenish the queue quickly after a branch has occurred.
- Instruction queue also mitigates the impact of cache misses:
  - In the event of a cache miss, the dispatch unit continues to send instructions to the execution unit as long as the queue is full.
  - In the meantime, the desired cache block is read.
  - If the queue does not become empty, cache miss has no effect on the rate of instruction execution.

Conditional branches and branch prediction

- Conditional branch instructions depend on the result of a preceding instruction.
  - Decision on whether to branch cannot be made until the execution of the preceding instruction is complete.
- Branch instructions represent 20% of the dynamic instruction count of most programs.
  - Dynamic instruction count takes into consideration that some instructions are executed repeatedly.
- Branch instructions may cause branch penalty reducing the performance gains expected from pipelining.
- Several techniques to reduce the negative impact of branch penalty on performance.

Delayed branch

- *Branch target address is computed in stage E2.*
- *Instructions I_3 and I_4 have to be discarded.*
- *Location following a branch instruction is called a branch delay slot.*
- *There may be more than one branch delay slot depending on the time it takes to determine whether the instruction is a branch instruction.*
- *In this case, there are two branch delay slots.*
- *The instructions in the delay slot are always fetched and at least partially executed before the branch decision is made and the branch address is computed.*



Delayed branch (contd..)

- Delayed branching can minimize the penalty incurred as a result of conditional branch instructions.
- Since the instructions in the delay slots are always fetched and partially executed, it is better to arrange for them to be fully executed whether or not branch is taken.
  - If we are able to place useful instructions in these slots, then they will always be executed whether or not the branch is taken.
- If we cannot place useful instructions in the branch delay slots, then we can fill these slots with NOP instructions.

Delayed branch (contd..)

```
LOOP      Shift_left     R1
          Decrement      R2
          Branch≠0       LOOP
NEXT      Add            R1,R3
```

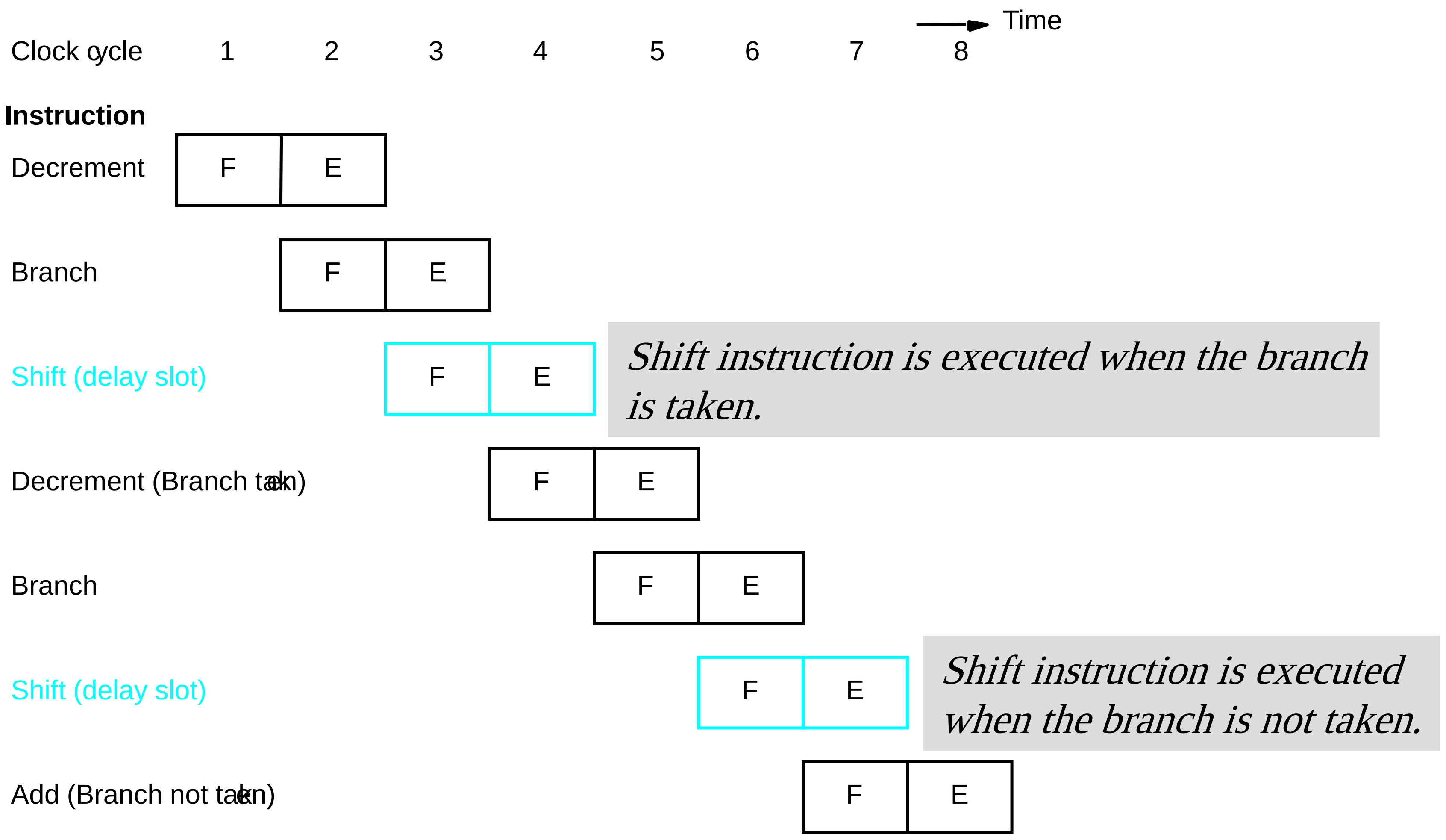
(a) Original program loop

```
LOOP      Decrement      R2
          Branch≠0       LOOP
          Shift_left     R1
NEXT      Add            R1,R3
```

(b) Reordered instructions

Register R2 is used as a counter to determine how many times R1 is to be shifted.
Processor has a two stage pipeline or one delay slot.
Instructions can be reordered so that the shift left instruction appears in the delay slot.
Shift left instruction is always executed whether the branch condition is true or false.

The sequence of events in the last two iterations (passes)of the loop.



Shift instruction is executed when the branch is taken.

Shift instruction is executed when the branch is not taken.

Execution timing showing the delay slot being filled during the last two passes through the loop in Figure 8.12*b*.

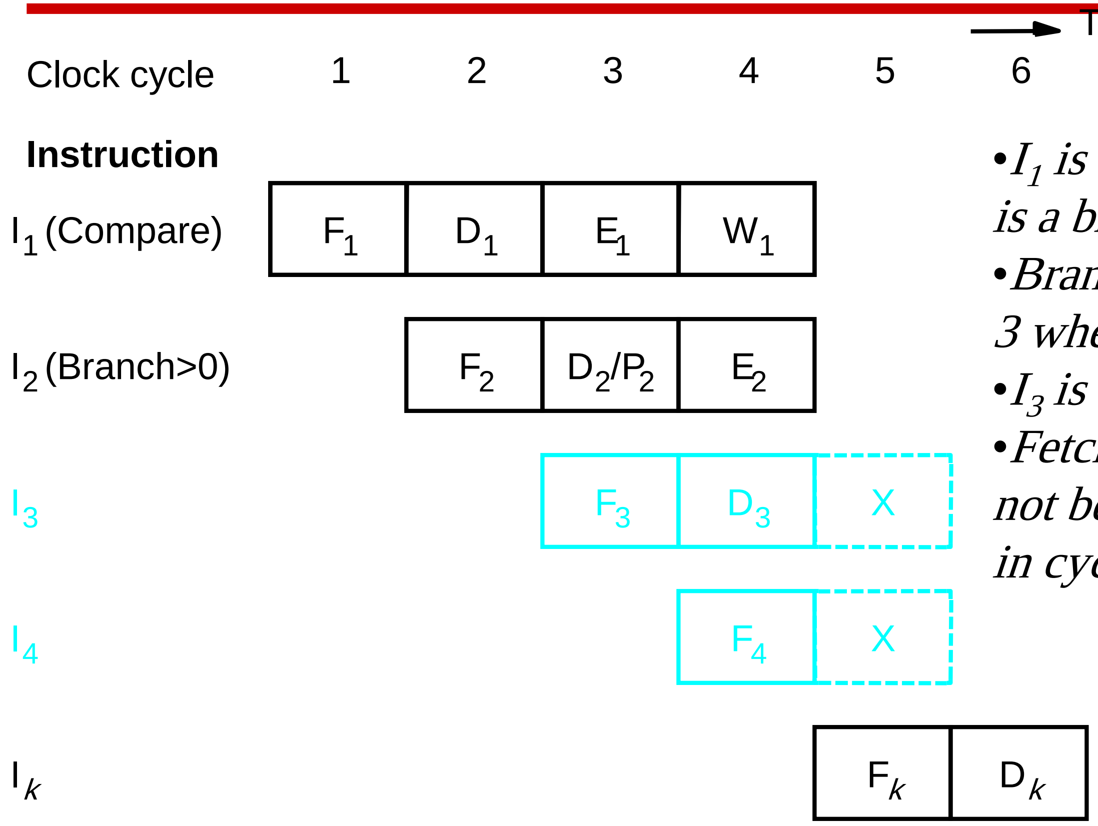
Delayed branch (contd..)

- Logically, the program is executed as if the branch instruction were placed after the shift instruction.
- Branching takes place one instruction later than where the branch instruction appears in the instruction sequence (with reference to reordered instructions).
- Hence, this technique is termed as "delayed branch".
- Delayed branch requires reordering as many instructions as the number of delay slots.
  - Usually possible to reorganize one instruction to fill one delay slot.
  - Difficult to reorganize two or more instructions to fill two or more delay slots.

Branch prediction

- To reduce the branch penalty associated with conditional branches, we can predict whether the branch will be taken.
- Simplest form of branch prediction:
  - Assume that the branch will not take place.
  - Continue to fetch instructions in sequential execution order.
  - Until the branch condition is evaluated, instruction execution along the predicted path must be done on a speculative basis.
- "Speculative execution" implies that the processor is executing instructions before it is certain that they are in the correct sequence.
  - Processor registers and memory locations should not be updated unless the sequence is confirmed.
  - If the branch prediction turns out to be wrong, then instructions that were executed on a speculative basis and their data must be purged.
  - Correct sequence of instructions must be fetched and executed.

Timing when a branch decision has been incorrectly predicted as not taken.



- *I_1 is a compare instruction and I_2 is a branch instruction.*
- *Branch prediction takes place in cycle 3 when I_2 is being decoded.*
- *I_3 is being fetched at that time.*
- *Fetch unit predicts that the branch will not be taken and continues to fetch I_4 in cycle 4 when I_3 is being decoded.*

- *Results of I_1 are available in cycle 3.*
- *Fetch unit evaluates branch condition in cycle 4.*
- *If the branch prediction is incorrect, the fetch unit realizes at this point.*
- *I_3 and I_4 are discarded and I_k is fetched from the branch target address.*

Branch prediction (contd..)

- If branch outcomes were random, then the simple approach of always assuming that the branch would not be taken would be correct 50% of the time.
- However, branch outcomes are not random and it may be possible to determine a priori whether a branch will be taken or not depending on the expected program behavior.
  - For example, a branch instruction at the end of the loop causes a branch to the start of the loop for every pass through the loop except the last one. Better performance can be achieved if this branch is always predicted as taken.
  - A branch instruction at the beginning of the loop causes the branch to be not taken most of the time. Better performance can be achieved if this branch is always predicted as not taken.

Branch prediction (contd..)

- Which way to predict the result of the branch instruction (taken or not taken) may be made in the hardware, depending on whether the target address of the branch instruction is lower or higher than the address of the branch instruction.
  - If the target address is lower, then the branch is predicted as taken.
  - If the target address is higher, then the branch is predicted as not taken.
- Branch prediction can also be handled by the compiler.
  - Complier can set the branch prediction bit to 0 or 1 to indicate the desired behavior.
  - Instruction fetch unit checks the branch prediction bit to predict whether the branch will be taken.

Branch prediction (contd..)

- Branch prediction decision is the same every time an instruction is executed.
  - This is "static branch prediction".
- Branch prediction decision may change depending on the execution history.
  - This is "dynamic branch prediction".

Branch prediction (contd..)

- Branch prediction algorithms should minimize the probability of making a wrong branch prediction decision.
- In dynamic branch prediction the processor hardware assesses the likelihood of a given branch being taken by keeping track of branch decisions every time that instruction is executed.
- Simplest form of execution history used in predicting the outcome of a given branch instruction is the result of the most recent execution of that instruction.
  - Processor assumes that the next time the instruction is executed, the result is likely to be the same.
  - For example, if the branch was taken the last time the instruction was executed, then the branch is likely to be taken this time as well.

Superscalar operation

- Pipelining enables multiple instructions to be executed concurrently by dividing the execution of an instruction into several stages:
  - Instructions enter the pipeline in strict program order.
  - If the pipeline does not stall, one instruction enters the pipeline and one instruction completes execution in one clock cycle.
  - Maximum throughput of a pipelined processor is one instruction per clock cycle.
- An alternative approach is to equip the processor with multiple processing units to handle several instructions in parallel in each stage.

Superscalar operation (contd..)

- If a processor has multiple processing units then several instructions can start execution in the same clock cycle.
  - Processor is said to use "multiple issue".
- These processors are capable of achieving instruction execution throughput of more than one instruction per cycle.
- These processors are known as "superscalar processors".
- Battery-powered devices, essentially all general-purpose CPUs developed since about 1998 are superscalar.
- The Intel Pentium (P5) was the first superscalar processor.

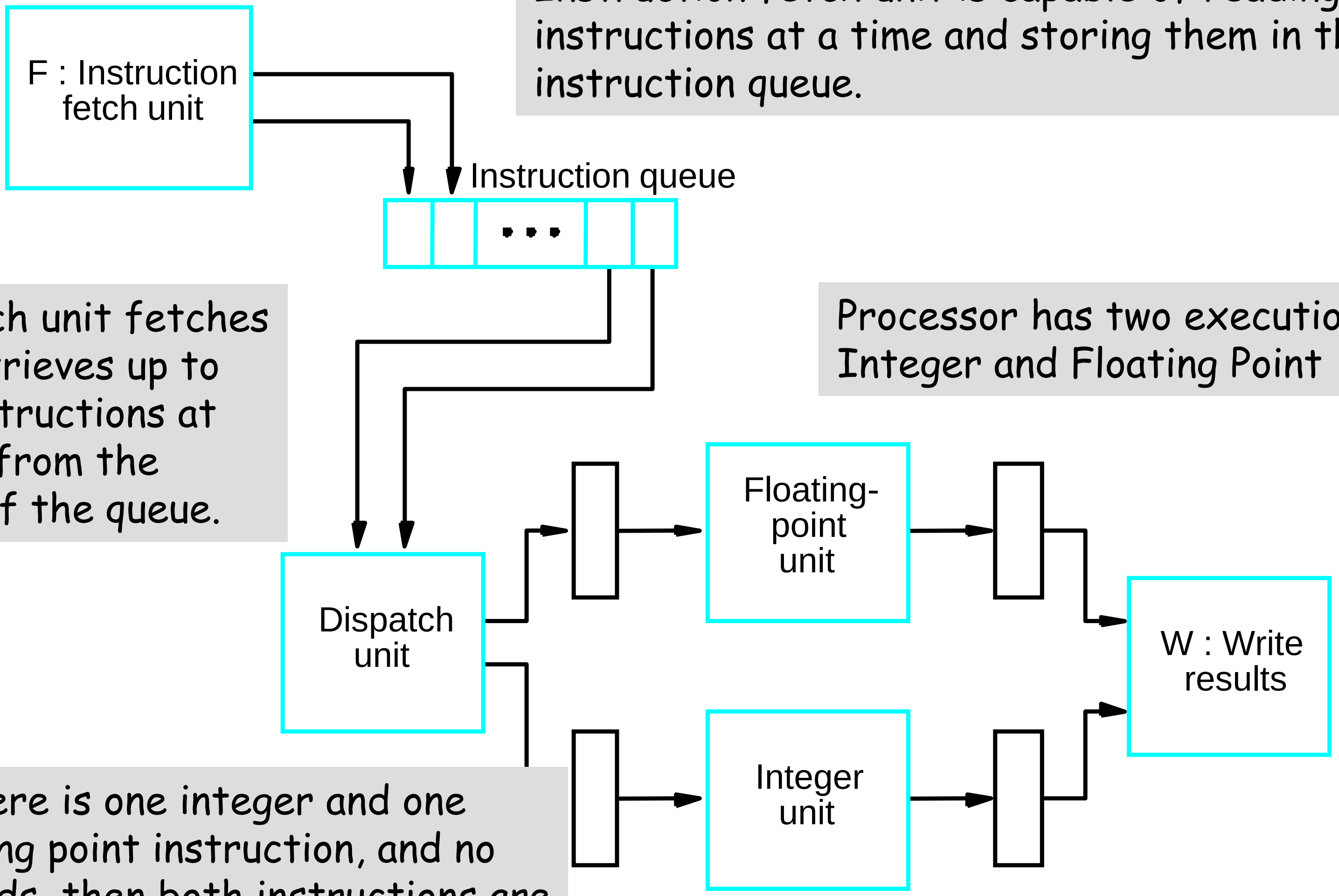
Superscalar operation (contd..)

Instruction fetch unit is capable of reading two instructions at a time and storing them in the instruction queue.

Dispatch unit fetches and retrieves up to two instructions at a time from the front of the queue.

Processor has two execution units: Integer and Floating Point

If there is one integer and one floating point instruction, and no hazards, then both instructions are dispatched in the same clock cycle.



Superscalar operation (contd..)

- Various hazards cause a even greater deterioration in performance in case of a superscalar processor.
- Compiler can avoid many hazards by careful ordering of instructions:
  - For example, the compiler should try to interleave floating-point and integer instructions.
  - Dispatch unit can then dispatch two instructions in most clock cycles, and keep both integer and floating point units busy most of the time.
- If the compiler can order instructions in such a way that the available hardware units can be kept busy most of the time, high performance can be achieved.

Superscalar operation (contd..)

Clock cycle	1	2	3	4	5	6	7
I_1 (Fadd)	F_1	D_1	E_{1A}	E_{1B}	E_{1C}	W_1	
I_2 (Add)	F_2	D_2	E_2	W_2			
I_3 (Fsub)		F_3	D_3	E_3	E_3	E_3	W_3
I_4 (Sub)		F_4	D_4	E_4	W_4		

- *Instructions in the floating-point unit take three cycles to execute.*
- *Floating-point unit is organized as a three-stage pipeline.*
- *Instructions in the integer unit take one cycle to execute.*
- *Integer unit is organized as a single-stage pipeline.*
- *Clock cycle 1:*
  - *Instructions I_1 (floating point) and I_2 (integer) are fetched.*
- *Clock cycle 2:*
  - *Instructions I_1 and I_2 are decoded and dispatched, I_3 is fetched.*

Superscalar operation (contd..)

Clock cycle	1	2	3	4	5	6	7
I_1 (Fadd)	F_1	D_1	E_{1A}	E_{1B}	E_{1C}	W_1	
I_2 (Add)	F_2	D_2	E_2	W_2			
I_3 (Fsub)		F_3	D_3	E_3	E_3	E_3	W_3
I_4 (Sub)		F_4	D_4	E_4	W_4		

- *Clock cycle 3:*
  - *I_1 and I_2 begin execution, I_2 completes execution. I_3 is dispatched to floating*
  - *point unit and I_4 is dispatched to integer unit.*

Clock cycle 4:
- *I_1 continues execution, I_3 begins execution, I_2 completes Write stage, I_4 completes execution.*

Clock cycle 5:
- *I_1 completes execution, I_3 continues execution, and I_4 completes Write.*

Order of completion is I_2, I_4, I_1, I_3

Question

What is the difference between
Pipelining,
Superscalar
and Multicore processor design.

RISC versus CISC Machines

Performance equation

$T=(N \times S) / R = N \times S \times P$

where T is the processor time required to execute a program, N is the number of instruction executions, and S is the average number of basic steps needed to execute one machine instruction. N, S, and R are not independent parameters.
the length P of one clock cycle, its inverse is the clock rate, $R=1/P$

RISC versus CISC Machines

RISC (Reduced Instruction Set Computing) and CISC (Complex Instruction Set Computing) are two computer architectures that are predominantly used nowadays.

The CISC Approach
The primary goal of CISC architecture is to complete a task in as few lines of assembly as possible(trying to reduce N). This is achieved by building processor hardware that is capable of understanding and executing a series of operations(complex operations).

The RISC Approach
RISC processors only use simple instructions that can be executed within one clock cycle (trying to reduce S and P) .

RISC versus CISC Machines

Aspect	RISC	CISC
Acronym	Reduced Instruction Set Computer	Complex Instruction Set Computer
Instruction set	Reduced(simple)	Complex
Compiler design	Hard to design	Easy to design
Hardware/Software	Stresses more on software	Stresses more on hardware
Code size	High in code size	Less in code size
Clocking	Single clock is used	Multiple clocks are used
Instruction length	Single word instruction	Variable length instruction
Pipelining	Pipelining is the major feature	Doesn't support pipelining
Memory	Can use more RAM to store intermediate results	Use less RAM as no need to store intermediate results

RISC has managed to work its way into portable devices like smartphones, tablets, and other similar devices. ARM is one of the notable RISC architectures used in these devices.

RISC versus CISC Machines

Regardless of the differences in them, both of the architectures are widely used today.
One of the reasons why Intel are still using CISC is because most of the current systems and software are x86standard.
X86 is a CISC standard, and switching to RISC means that all the previous systems would no longer run on the new architectures. However, RISC chips have their own success stories: RISC processors have been used in Apple iPhones, iPads, Blackberrys and in many more devices.
Usually X86 (or X32) refers to 32-bit software versions while X64 refers to 64-bit software versions.
64-bit systems can run 32-bit programs, as they're backwards compatil
It doesn't work the other way around, though: a 32-bit computer cannot run 64-bit Windows or 64-bit programs

End